Recommending Related Functions from API Usage-Based Function Clone Structures*

Shamsa Abid
Lahore University of Management Sciences
Lahore, Pakistan
15030049@lums.edu.pk

ABSTRACT

Developers need to be able to find reusable code for desired software features in a way that supports opportunistic programming for increased developer productivity. Our objective is to develop a recommendation system that provides a developer with function recommendations having functionality relevant to her development task. We employ a combination of information retrieval, static code analysis and data mining techniques to build the proposed recommendation system called FACER (**F**eature-driven **A**PI usage-based **C**ode **E**xamples **R**ecommender). We performed an experimental evaluation on 122 projects from GitHub from selected categories to determine the accuracy of the retrieved code for related features. FACER recommended functions with a precision of 54% and 75% when evaluated using automated and manual methods respectively.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Software and its engineering** → **Software libraries and repositories**; **Search-based software engineering**.

KEYWORDS

code recommendation, software features, API usage, code clones

ACM Reference Format:
Shamsa Abid. 2019. Recommending Related Functions from API Usage-Based Function Clone Structures. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3338906.3342486

1 INTRODUCTION

Developers need to be able to find reusable code for the desired features in a way that supports opportunistic programming [5], which is an iterative process of adding features to code under development. Current code search and recommender systems [2, 6, 9, 14, 15, 17, 25, 35] focus on providing code against a specific user query and repeated searches need to be performed until the code for the desired feature or set of features is found. The problem of repeated code searches needs to be addressed to improve developer's productivity and facilitate rapid application development (RAD). Existing feature recommendation systems [7, 8, 20, 33, 34] enable the exploration of related features for RAD and domain analysis. However, they do not provide code at a fine-grained function level for reuse. A solution is desired for the recommendation of related functions for reuse in an opportunistic manner.

1.1 Motivating Example

Consider the example of an Android music player application. Suppose the developer wants to implement a feature that allows her to play a media file. For example, if the developer queries for the feature "play media file", then along with the code needed to implement playing a media file, code for related features like "pause media file", "get current track progress", "check if media playing" and "handle touch event" will also help.

2 BACKGROUND AND RELATED WORK

Various code search and recommendation systems have been proposed in the literature [2, 6, 9–11, 13–17, 21–32, 35]. However, only a few predict and recommend related code that may be useful in the later stages of software development. In this direction, McMillan et al. [20] proposed an approach that identifies and recommends features and associated Java packages that are related to the software under development. *Portfolio* [19] retrieves relevant functions against a user query, however, the related functions are limited to the same call graph. Ichii et al. [12] and *Rascal* [18] use collaborative filtering to recommend code based on browsing session and class code similarities respectively. The last two approaches will only work if a developer history or code profile is available.

3 APPROACH

Figure 1 (a,b) shows the architecture of FACER. In the off-line repository building process (Figure 1 (b)), we use static code analysis for extracting search facets (comments and keywords), function call graphs and API usages from open source Java projects to construct a code fact repository. The Program Analyzer extracts comments of a function, keywords from the function name and API class names & API method names occurring within that function. These search facets are then preprocessed by splitting any identifiers to terms, using Camel-case. The resultant set of terms is then indexed using the open-source information retrieval engine Lucene [1]. The Program Analyzer also parses function calls and API usages occurring within a function and stores the data in the FACER repository. We then perform API usage-based semantic Function Clone Structures[1]

*This work was funded through Ignite Research Grant No. SRG-257.

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia

ACM ISBN 978-1-4503-5572-8/19/08.
https://doi.org/10.1145/3338906.3342486

[1] A Function Clone Structure (FCS) consists of a group of functions, which are cloned across different projects. Such a repeating group of function clones is called a Function Clone Structure.

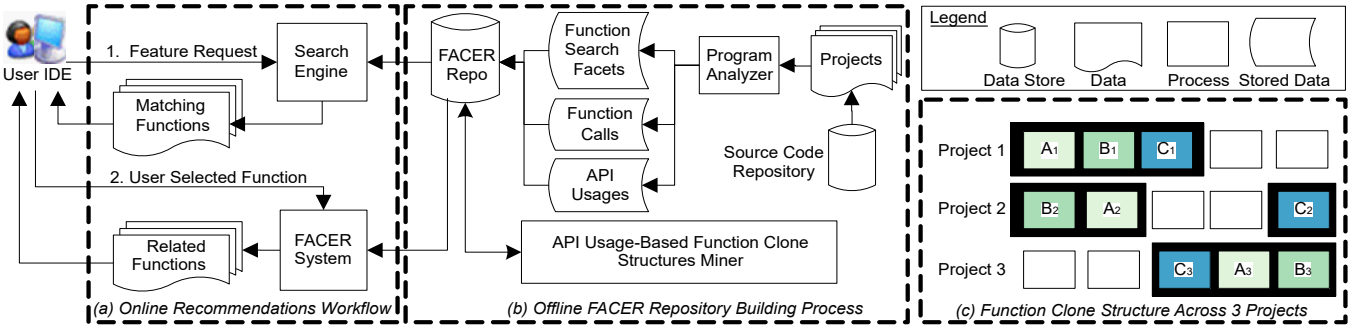

Figure 1: FACER System Architecture with Function Clone Structure of Repeatedly Co-occurring Function Clones

detection to learn the co-occurring functionality or related features across these projects and save this information in the same repository. This involves clustering functions with similar API usages into clone classes[2] and then mining frequent patterns of co-occurring clone classes to get the Function Clone Structures [3, 4]. In the online recommendations workflow (Figure 1 (a)), a developer's feature query (expressed as a comment) triggers the first recommendation of providing a set of functions that implement the feature. Upon selection of one of these functions by the developer, a subsequent recommendation provides code of the related functions for reuse. The clone structure(s) of the user selected function is/are determined and representative functions of the clone classes found in the structure(s) are recommended as related functions. Figure 1 (c) shows functions present in projects 1, 2 and 3. Three function clone classes A, B and C are found across the projects. A_1 denotes function of project 1 belonging to clone class A. A_i, B_i and C_i together form a function clone structure where members of the clone structure are all related to each other.

4 EVALUATION AND RESULTS

The data source for the evaluation consists of 122 Java based Android applications from five different categories, downloaded from GitHub based on higher star rating. The categories include music player, Bluetooth chat, weather, file management and games. Among these, 101 projects were selected to build the FACER code repository. The remaining 21 projects formed the holdout set used for automated evaluation. The FACER repository includes 6355 files, 204603 lines of comments, 761053 lines of code, 66719 functions, 240311 function calls, 130984 API usages, 6354 clone classes and 711 function clone structures.

A set of ten candidate queries was prepared using the feature descriptions of a random project from each category. Once a query is executed on the system, a set of top 10 matching functions is retrieved against it. Starting from the top, each function is analyzed until a match is found. Once a candidate function is selected, more functions are automatically retrieved as code for related features against the feature query. The recommended functions are judged by a human expert to determine relevance based on her experience.

[2]If a group of functions share similar API usages (at least 50%), they are classified as API usage-based function clones belonging to a unique clone class. These functions implement the same feature or functionality and are instances of that particular feature.

Table 1: Automated and Manual Evaluation Precision Values (R=Relevant, T=Total, P=Precision)

No.	Query	R	T	$P_m = R/T$	P_a
1	Play music file	1	1	1.00	0.60
2	Connect to a Bluetooth device	7	7	1.00	0.83
3	Get weather data	3	3	1.00	0.27
4	Compress file	16	19	0.84	0.54
5	Open file	6	9	0.66	0.57
6	Send message to Bluetooth device	4	4	1.00	0.53
7	Create playlist of music files	22	30	0.73	0.52
8	View list of songs	12	22	0.54	0.53
9	Browse file directory	9	12	0.75	0.51
10	Free memory kill tasks	0	6	0.00	0.50
	Average Precision			**0.75**	**0.54**

For automated evaluation, the precision of the recommended functions is calculated by comparing their API usages against the API usages of holdout projects. Table 1 shows the manual precision (P_m) and automated precision (P_a) of related function recommendations evaluated using manual and automated methods for each query. The average precision using manual evaluation is 75% whereas it is 54% using automated evaluation. Manual evaluation indicated a high precision. In contrast, the precision reported in literature by the state-of-the-art feature recommender [20] is 59% through manual evaluation. Some of the recommendations that our system made against the fourth query "Compress file" were "Check files", "Delete files", "Copy file to a new directory", "Search file", "Create zip file" and "Rename a file".

5 CONCLUSION AND CONTRIBUTIONS

We introduce the notion of API usage similarity between functions as the basis of semantic function clones and propose a technique to discover API usage-based function clone classes and structures. Based on this technique we recommend functions related to the users' desired feature request in an opportunistic fashion. We implemented the above technique in a tool called FACER and conducted experiments to demonstrate the use of API usage-based function clone structures to recommend related functions with a manual precision of 75% and automated precision of 54%.

REFERENCES

[1] [n.d.]. https://lucene.apache.org. [Online; accessed 29-June-2019].
[2] Sushil K Bajracharya, Joel Ossher, and Cristina V Lopes. 2010. Leveraging usage similarity for effective retrieval of examples in code repositories. In *Proceedings of*

the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering. ACM, 157–166.

[3] Hamid Abdul Basit, Usman Ali, Sidra Haque, and Stan Jarzabek. 2012. Things structural clones tell that simple clones don't. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 275–284.

[4] Hamid Abdul Basit and Stan Jarzabek. 2009. A data mining approach for detecting higher-level clones in software. *IEEE Transactions on Software engineering* 35, 4 (2009), 497–514.

[5] Joel Brandt, Philip J Guo, Joel Lewenstein, and Scott R Klemmer. 2008. Opportunistic programming: How rapid ideation and prototyping occur in practice. In *Proceedings of the 4th international workshop on End-user software engineering*. ACM, 1–5.

[6] Shaunak Chatterjee, Sudeep Juvekar, and Koushik Sen. 2009. Sniff: A search engine for java using free-form queries. *Fundamental Approaches to Software Engineering* (2009), 385–400.

[7] Xiangping Chen, Qiwen Zou, Bitian Fan, Zibin Zheng, and Xiaonan Luo. 2018. Recommending software features for mobile applications based on user interface comparison. *Requirements Engineering* (2018), 1–15.

[8] Horatiu Dumitru, Marek Gibiec, Negar Hariri, Jane Cleland-Huang, Bamshad Mobasher, Carlos Castro-Herrera, and Mehdi Mirakhorli. 2011. On-demand feature recommendations derived from mining public product descriptions. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 181–190.

[9] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 933–944.

[10] Reid Holmes, Robert J Walker, and Gail C Murphy. 2005. Strathcona example recommendation tool. In *ACM SIGSOFT Software Engineering Notes*, Vol. 30. ACM, 237–240.

[11] Oliver Hummel, Werner Janjic, and Colin Atkinson. 2008. Code conjurer: Pulling reusable software out of thin air. *IEEE software* 25, 5 (2008).

[12] Makoto Ichii, Yasuhiro Hayase, Reishi Yokomori, Tetsuo Yamamoto, and Katsuro Inoue. 2009. Software component recommendation using collaborative filtering. In *2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*. IEEE, 17–20.

[13] Tomoya Ishihara, Keisuke Hotta, Yoshiki Higo, and Shinji Kusumoto. 2013. Reusing reused code. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*. IEEE, 457–461.

[14] He Jiang, Liming Nie, Zeyi Sun, Zhilei Ren, Weiqiang Kong, Tao Zhang, and Xiapu Luo. 2016. Rosf: Leveraging information retrieval and supervised learning for recommending code snippets. *IEEE Transactions on Services Computing* (2016).

[15] Iman Keivanloo, Juergen Rilling, and Ying Zou. 2014. Spotting working code examples. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 664–675.

[16] Sifei Luan, Di Yang, Koushik Sen, and Satish Chandra. 2018. Aroma: Code Recommendation via Structural Code Search. *arXiv preprint arXiv:1812.01158* (2018).

[17] Fei Lv, Hongyu Zhang, Jian-guang Lou, Shaowei Wang, Dongmei Zhang, and Jianjun Zhao. 2015. Codehow: Effective code search based on api understanding and extended boolean model (e). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 260–270.

[18] Frank Mccarey, Mel Ó Cinnéide, and Nicholas Kushmerick. 2005. Rascal: A recommender agent for agile reuse. *Artificial Intelligence Review* 24, 3-4 (2005), 253–276.

[19] Collin McMillan, Mark Grechanik, Denys Poshyvanyk, Qing Xie, and Chen Fu. 2011. Portfolio: finding relevant functions and their usage. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 111–120.

[20] Collin McMillan, Negar Hariri, Denys Poshyvanyk, Jane Cleland-Huang, and Bamshad Mobasher. 2012. Recommending source code for use in rapid software prototypes. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 848–858.

[21] Alon Mishne, Sharon Shoham, and Eran Yahav. 2012. Typestate-based semantic code search over partial programs. In *Acm Sigplan Notices*, Vol. 47. ACM, 997–1016.

[22] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Andrian Marcus. 2015. How can I use this method?. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, Vol. 1. IEEE, 880–890.

[23] Mukund Raghothaman, Yi Wei, and Youssef Hamadi. 2016. SWIM: Synthesizing What I Mean-Code Search and Idiomatic Snippet Synthesis. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 357–367.

[24] Steven P Reiss. 2009. Semantics-based code search. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 243–253.

[25] Saksham Sachdev, Hongyu Li, Sifei Luan, Seohyun Kim, Koushik Sen, and Satish Chandra. 2018. Retrieval on Source Code: A Neural Code Search. In *Proceedings of the 2Nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL 2018)*. ACM, New York, NY, USA, 31–41. https://doi.org/10.1145/3211346.3211353

[26] Naiyana Sahavechaphan and Kajal Claypool. 2006. XSnippet: Mining for sample code. *ACM Sigplan Notices* 41, 10 (2006), 413–430.

[27] Ryuji Shimada, Yasuhiro Hayase, Makoto Ichii, Makoto Matsushita, and Katsuro Inoue. 2009. A-SCORE: Automatic software component recommendation using coding context. In *2009 31st International Conference on Software Engineering-Companion Volume*. IEEE, 439–440.

[28] Watanabe Takuya and Hidehiko Masuhara. 2011. A spontaneous code recommendation tool based on associative search. In *Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*. ACM, 17–20.

[29] Jue Wang, Yingnong Dang, Hongyu Zhang, Kai Chen, Tao Xie, and Dongmei Zhang. 2013. Mining succinct and high-coverage API usage patterns from source code. In *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 319–328.

[30] Lijie Wang, Lu Fang, Leye Wang, Ge Li, Bing Xie, and Fuqing Yang. 2011. APIExample: An effective web search based usage example recommendation system for Java APIs. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 592–595.

[31] Tao Xie and Jian Pei. 2006. MAPO: Mining API usages from open source repositories. In *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 54–57.

[32] Yunwen Ye and Gerhard Fischer. 2002. Supporting reuse by delivering task-relevant and personalized information. In *Proceedings of the 24th international conference on Software engineering*. ACM, 513–523.

[33] Hong Yu, Yahong Lian, Shuotao Yang, Linlin Tian, and Xiaowei Zhao. 2016. Recommending features of mobile applications for developer. In *International Conference on Advanced Data Mining and Applications*. Springer, 361–373.

[34] Yue Yu, Huaimin Wang, Gang Yin, and Bo Liu. 2013. Mining and recommending software features across multiple web repositories. In *Proceedings of the 5th Asia-Pacific Symposium on Internetware*. ACM, 9.

[35] Alexey Zagalsky, Ohad Barzilay, and Amiram Yehudai. 2012. Example overflow: Using social media for code recommendation. In *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*. IEEE Press, 38–42.